基于容器技术的天文应用软件自动部署方法*

姚坤[1]，戴伟[1*]，杨秋萍[1]，梅盈[23]，石聪明[1]，王锋[123]
(1,昆明理工大学云南省计算机技术应用重点实验室,云南 昆明,650500; 2,中国科学院云南天文台,云南 昆明,650011; 3,广州大学,天体物理中心,广东 广州,510006)

摘要：平方公里阵望远镜(SKA)即将开始建设，各子工作包也将进入关键设计评估阶段。基于云与容器技术是当前 SKA 科学数据处理器（SDP）未来可能采用的平台技术。本文针对 SDP 超大规模海量数据处理中所面临的天文应用软件快速部署、运行与实测要求，充分考虑了天文应用软件运行环境复杂，云计算环境下超大规模计算集群部署困难等问题，系统研究并给出了一种使用 Docker 技术的天文应用软件通用自动部署方法，论文以目前较为常用的可见度函数校准软件 SAGECaL 为例，首先分析了 SAGECaL 的相关特性和分布式部署方面存在的困难，进而给出了基于 Docker 容器技术的 SAGECaL 分布式集群的自动部署方法。实验结果表明，本文所提出的自动部署方法极大地提高了 SAGECaL 分布式集群的部署效率，满足了项目组承担 SKA-SDP 相关测试工作中所需要的基础平台部署与切换等需求。同时，本工作也为其它天文软件在云端的快速部署与执行提供了有益的思路。

关键词：自动部署；容器技术；SAGECaL；SKA；SDP

中图分类号：TP311.1　**文献标识码**：A　**文章编号**：1672-7673(2018)xx-xxxx-xx

0 引 言

平方公里阵（SKA）望远镜是目前待建的全世界最大口径天文射电望远镜，其中科学数据处理器（SDP）是 SKA 重要的工作包之一。为了解决 SDP 在超大规模数据处理中所面临的问题，近几年科学家们在各个领域都开展了一系列研究，包括从硬件加速器（GPU 等）、计算框架[1]、系统框架[2]等，取得了一系列的成果[3]。

在诸多平台技术选型中，云计算以节省成本、资源动态整合、动态可扩展、简化管理维护和灾难恢复等特性成为了 SKA 天文数据处理的一种选择。与一般的基于互联网的应用部署不同，天文软件一般在部署运行时需要非常多的三方库支持，运行时也会有诸多的参数以满足不同的要求。天文领域对容器技术的使用还停留在天文应用软件的封装层面。特别是在天文应用软件有专人管理的情况下，用户还面临等待系统管理员部署特定软件的交付延迟问题[4]。如何在云环境下快速部署，大幅度降低使用成本和提高整体系统的可用性迫切需要进一步实验与研究。同时，如何使用 Docker[5]配合 OpenStack[6]、Mesos 和 Kubernates 等容器编排软件来对复杂的天文应用软件进行封装从而加快部署速度[7] 正在成为当前天文应用软件的热点。

SAGECaL(Space Alternating Generalized Expectation Maximization Calibration)[8]是一种执行速度快、内存效率高的射电干涉校准程序。它支持点源、高斯和 Shapelet 等多种模型。SAGECal 支持 CASA 的 MS（Measurement Set）数据，在其分布式应用中实现了 ADMM（Alternating Direction Method of Multipliers）参数的自适应更新[9]。SAGECaL 使用期

* 基金项目：国家重点研发计划（2018YFA0404603，2016YFE0100300），国家自然科学基金委员会-中国科学院天文联合基金资助重点项目（U1831204），国家自然科学基金委员会-中国科学院天文联合基金资助项目(U1531132,U1631129)，国家自然科学基金资助项目(No.11403009,11463003,11773012)，云南省应用基础研究项目(2017FB001)，赛尔网络下一代互联网技术创新项目（No.NGII20170204）资助。

作者简介：姚坤，男，硕士研究生.研究方向：计算机应用技术.Email:9259813@qq.com
通讯作者：戴伟，男，副教授.研究方向：计算机应用技术.Email：dw@cnlab.net

望最大化（Expectation-Maximization，EM）算法获得对于观测仪器和天空模型的最大似然估计参数的技术。普通的 EM 算法应用非线性叠加信号的参数估计，SAGECaL 采用一种被称为 SAGE 算法的 EM 改进算法来达到与普通 EM 算法相比更快的收敛速度；与最小二乘估计方法相比减少了计算复杂度；与剥离校准方法相比提高了校准质量。为了应对新一代射电干涉望远镜（如 SKA）的海量数据计算需求，SAGECaL 支持在频率上分布式并行校准，代码实现时可以支持单 CPU、硬件 GPU 或 MPI 并行计算等多种部署环境。

在 SKA-SDP 方案设计阶段，为给出较为可信的计算资源需求分析以及能耗设计，项目组和国外团队在合作中，拟通过在不同环境下 SAGECal 的部署与实际运行，给出不同配置、不同节点数、不同参数和输出精度情况下的系统开销分析结果，根据能源与计算开销最终给出 SDP 数据处理的较优设计。显然，采用传统的手工安装与环境配置的方法远远满足不了工作开展的要求。本文在云环境下快速部署 SAGECal 软件开展了深入研究，获得了较好的解决方法，有效推动了 SDP 相关工作的开展。

1 基于容器技术的自动部署

1.1 基本需求分析

为了有效地给出测试数据，我们重点研究了如何快速部署一个基于 SAGECaL/MPI 的分布式集群（以下简称集群）。下面列出了部署过程中的关键点：

- 根据节点情况编辑集群列表
- 设置集群节点处于同一网段
- 配置 NFS
- 配置各节点 SSH（Secure Shell）免秘钥登录
- 配置 MPI 集群环境
- 各节点的 CASACORE、SAGECaL 和 OpenMPI 使用同一版本编译

在实际部署和使用过程中面还面临着以下问题：(1) 由于用户硬件环境和使用偏好的差异使得用户所选 Linux 操作系统发行版本呈现多样化，SAGECaL 的编译和部署需要进行专有设置；(2) SAGECaL 的编译需要人工解决组件依赖关系，特别是 SAGECaL-MPI 和 SAGECaL-GPU 的编译，还需根据实际情况人工编辑配置文件才能编译成功。这对于非专业人员来说，极大地增加了部署的难度。

针对上述关键点，拟通过一套配置脚本自动配置部署过程中的每一个环节。在实际部署和使用过程中面临的问题可以通过 Docker 对 SAGECaL 进行封装来解决。配合使用交互方式友好的用户接口将进一步降低使用难度。

1.2 部署的整体思路

集群的自动部署是通过 Docker 构造 SAGECal 镜像（以下简称镜像）并通过部署服务启动镜像生成多个 SAGECaL 容器（以下简称容器），然后各容器通过 Overlay 网络进行跨容器通讯来实现自动配置。其中，构造镜像时不仅封装了 SAGECal 软件，还封装了自动部署的配置脚本用以启动节点时根据节点角色的不同产生不同的配置行为。Docker Swarm（以下简称 Swarm）是 Docker 自带的集群管理工具，其主要作用是在若干台主机上建立容器云并通过统一入口管理容器云上的各种资源[10]。

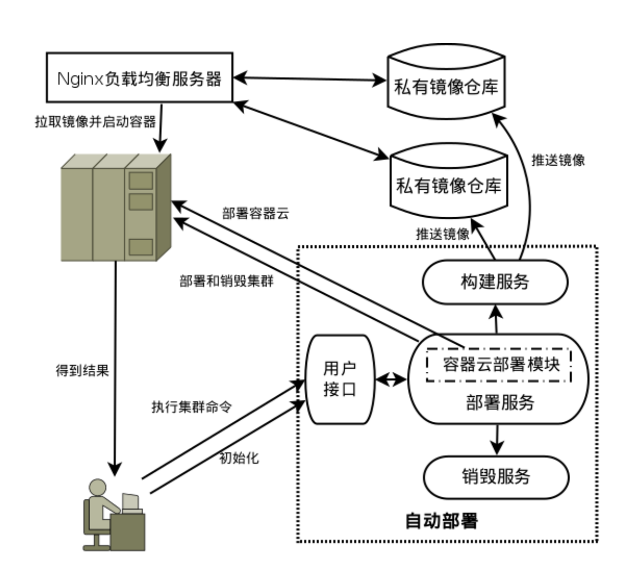


图 1 自动部署流程图

Fig.1 Diagram of automatic deployment

用户首先对可用机器进行初始化，包括创建 Swarm 管理的容器云、配置 NFS（Network File System）和创建私有镜像仓库。构建服务和销毁服务分别用于构建镜像和集群销毁。部署服务会调用构建服务和销毁服务并实现了服务编排、弹性伸缩、集群节点发现与维护和用户接口等功能。在集群运算结束后用户可以从 NFS 指定目录获取处理结果。

1.3 镜像的构建与组成

镜像是基于 Linux 主流发行版基础镜像构建的。本文选用 Debian8.6 基础镜像通过 Dockerfile 构建方式构建镜像，镜像中的主要软件包括 SSH、CASACORE、Glib、OpenBLAS、Gcc、Make、OpenMPI 和 SAGECal。

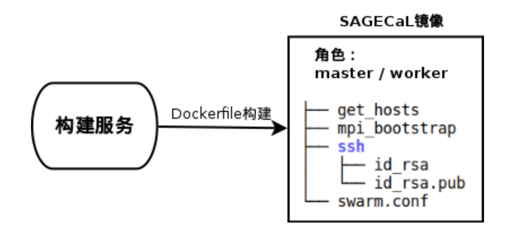


图 2 自动部署脚本封装示意图

Fig.2 Diagram of automatic deployment script encapsulation

作为通用镜像，为了让节点容器在启动后能够根据自身在集群中的不同角色自动进行不同的操作，在制作镜像的时候需要将定义了不同操作的脚本以及自动维护集群列表的相关脚本一同打包进镜像[11]。同样，为了实现集群内部节点间及裸机操作系统与集群主节点间的 SSH 免秘钥登录，裸机操作系统的公匙也需写入镜像并配置 SSH 登录免提示等相关配置工作。

1.4 自动部署

1.4.1 初始化

首先，对可用物理机进行初始化。提前选取一台物理机作为物理机机群的主节点，其余物理机作为物理机机群的子节点。通过人工配置的方式在各台物理机操作系统中配置 SSH

免秘钥登录和设置开机启动的自动化脚本使得各台物理机开机后会主动向物理机机群的主节点发起并保持 SSH 连接从而让主节点感知到其余物理机的存在并一直监控和维护物理机机群列表的变化（具体原理与 2.4.3 部分相同）。

其次，配置 NFS 用于容器间共享数据。在本文中所有机器都将各自的 NFS 共享目录挂载进容器，这样用户就可以通过裸机操作系统直接和容器进行数据交换。

然后，创建私有镜像仓库并配置负载均衡服务用于镜像分发。如图 1 所示，基于对镜像分发容错的考虑，私有镜像仓库采用横向扩展的形式分布在多台物理机上。构建服务构建好镜像后将同时推送镜像至各私有镜像仓库中。通过配置 Nginx 负载均衡服务以提供统一的镜像拉取地址供自动部署服务自动从私有镜像仓库下载所需镜像并启动容器进入工作状态。

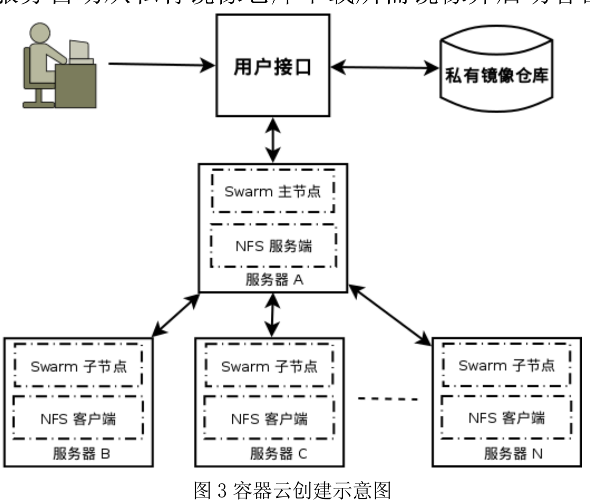


图 3 容器云创建示意图

Fig.3 Block diagram of container cloud creation

最后，自动部署容器云。容器云部署模块封装了直接在物理机机群主节点执行命令的脚本、按照 IP 列表批量执行命令的脚本和物理机机群列表对比监控等功能的脚本。如图 1 和图 3 所示，用户通过用户接口调用容器云部署模块对容器云进行自动部署。具体部署流程如下：

1、 容器云部署模块通过封装了 Swarm 主节点初始化命令的命令对 Swarm 主节点进行初始化并将获得的加入命令写入一个临时文件以备后期调用。
2、 容器云部署模块从物理机机群列表第二行开始依次读取 IP 地址并配合批量执行命令的脚本对目标物理机远程执行第一步中获得的加入 Swarm 集群的命令。
3、 物理机机群列表对比监控功能会持续通过封装了 Swarm 集群节点查看命令的功能模块维护 Swarm 集群 IP 列表，同时不断地将 Swarm 集群 IP 列表与物理机机群列表进行比对。当两个列表的行数一致、内容一致和 Swarm 节点的可达性为 Active 时，认为所有物理机均加入了 Swarm 集群，然后启动初始界面等待用户输入集群自动部署命令。
4、 当有新的物理机开机时物理机机群列表会被自动更新，同时触发容器云部署模块对新加入的物理机执行第一步中获得的加入命令使其自动加入 Swarm 集群。

通过物理机机群列表对比监控功能的持续监控，不但实现了物理机数量动态变化的容器云自动维护，还对容器云的自动部署具备一定的容错功能。后期的集群自动部署都将在容器云上实现。

1.4.2 创建 Overlay 网络和集群节点

在启动集群节点前，需要准备好 Overlay 网络以供节点容器在私有逻辑网上进行跨主机

通讯。在自动部署过程中创建 Overlay 网络是通过封装如下命令实现网络配置和指定网络名称。

```
docker network create                  \
       --driver overlay                \
       --subnet ${NETWORK_SUBNET}      \
       --opt encrypted                 \
       ${NETWORK_NAME}
```

自动部署还封装了 Swarm services 声明性模型的相关命令来定义服务的期望状态并依赖 Docker 来维持该状态和编排相关的服务。本文中集群的节点分别由主服务编排主节点，工作服务编排子节点。具体命令如下：

```
docker service create                                                   \
    --name ${MPI_MASTER_AUTOMATIC_SERVICE_NAME}                         \
    --replicas 1                                                        \
    --network ${NETWORK_NAME}                                           \
    --user root                                                         \
    --mount type=bind,source=$(pwd)/share/project,destination=/root/project \
    "${IMAGE_TAG}" mpi_bootstrap                                        \
            mpi_master_service_name=${MPI_MASTER_AUTOMATIC_SERVICE_NAME}\
            mpi_worker_service_name=${MPI_WORKER_AUTOMATIC_SERVICE_NAME}\
            host_number=${HOST_NUMBER}                                  \
            role=master-automatic
```

```
docker service create                                                   \
    --name ${MPI_WORKER_AUTOMATIC_SERVICE_NAME}                         \
    --replicas ${NUM_WORKER}                                            \
    --network ${NETWORK_NAME}                                           \
    --user root                                                         \
    --mount type=bind,source=$(pwd)/share/project,destination=/root/project \
    "${IMAGE_TAG}" mpi_bootstrap                                        \
            mpi_master_service_name=${MPI_MASTER_AUTOMATIC_SERVICE_NAME}\
            mpi_worker_service_name=${MPI_WORKER_AUTOMATIC_SERVICE_NAME}\
            role=worker-automatic
```

按照角色的不同，集群的节点起初都是每组服务启动一个容器，但子节点数会按照工作服务的配置进行弹性扩容。两组服务都绑定了同一个 Overlay 网络和一致的 NFS 目录。mpi_bootstrap 文件定义了一组操作命令，它会根据 *role* 参数决定这个节点容器在启动后执行何种操作。其中，主节点还传递了 *host_number* 参数作为集群规模的预设值。Swarm services 虽然可以设置容器数量，但运行集群还需要通过在容器内自动维护集群列表并确保达到集群规模预设值后才能执行用户输入的集群计算命令。

1.4.3 集群节点的发现与维护

集群节点的发现采用节点主动暴露 IP 的方式实现。主节点最先启动，启动后就开启对 ARP（Address Resolution Protocol）缓存的持续监控。各子节点启动后会主动向主节点发起 SSH 连接并访问主节点 dev 目录下的 null 文件。通过持续保持与 null 文件的连接，主节点便通过 ARP 缓存感知到子节点的 IP 变化以达到集群列表动态维护。SSH 连接命令如下：

```
ssh -T -o "StrictHostKeyChecking no"   \
    -i "/root/.ssh/id_rsa"             \
    root@${MPI_MASTER_SERVICE_NAME}    \
    "tail -f /dev/null"
```

由于预设了集群规模，所以还需要一个变量作为标识来通知集群维护模块所创建的集群是否达到既定规模。本文设置了 *CLUSTER_GET_READY* 变量并默认 0 为集群未准备就绪，1 为集群准备完毕。主节点通过不断监控集群列表的变化情况发现集群达到既定规模的时候，变量值被修改为 1 并告知主节点停止对集群列表的监控，然后启动图 5 所示初始界面等待用户输入集群计算命令。

1.4.4 用户接口

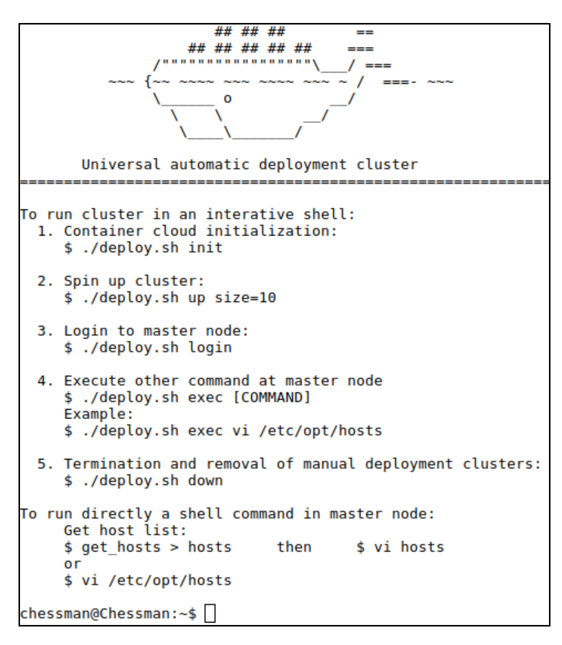


图 4 初始界面

Fig.4 Initial interface

集群的自动部署由图 4 中例 2 所示命令启动。用户根据实际需要设置集群规模，待自动部署完成后用户界面会停留在初始界面并等待用户输入集群计算命令或其他命令。通过提供交互友好的用户接口，用户只需提供较少的必要参数即可完成集群计算、终止并删除集群和进入集群节点查看运行情况等多种操作。

1.4.5 容错处理

集群自动部署过程中会出现硬件资源不足、主节点没有启动或崩溃和网络中断等因素导致部署过程出错。如果集群长时间没有准备就绪，自动部署模块会进行回退处理释放资源。如果回退失败，可以手工运行图 4 中例 5 所示命令强制销毁已创建的服务并将所占资源全部释放。经过对资源申请参数和网络可达性的进行检查后可再次执行集群自动部署命令进行自动部署。

总的来说，对于自动部署的全过程中如果各个部分的容错机制失效，采取的默认策略是进行回退并释放资源。

2 实验设计

上述步骤给出了以 SAGECaL 为例的自动部署方法。本章通过实验，测试并比较在不同服务器数量和不同集群规模情况下集群部署和集群计算的时效性以及选用分布式部署进行集群计算的合理性。

2.1 实验环境

实验所用服务器型号均为曙光天阔 620R，使用 Debian9.4 stable 作为物理机操作系统并安装 Docker18.03.1-ce 和 NFS 组件。基础服务节点安装了 Docker、NFS 服务端和部署 Docker 私有镜像仓库，其他节点安装了 Docker 和 NFS 客户端。镜像基于 Debian8.6 基础镜像采用 Dockerfile 构建镜像的方式进行构建。

本次实验使用十二台服务器来进行实验。其中基础服务节点作为主节点，其余机器作为子节点。服务器间使用千兆以太网相互连接且在同一个网段内且可以相互连通，集群运行在私有 Overlay 网络中。为了尽可能的还原真实的部署情形，每次进行实验之前都会清除服务器上所有镜像以确保每次自动部署都是从头执行。为了防止集群计算的时间被缓存所影响，每次实验完成后均对所有服务器进行重启操作，确保清空所有计算过程中所产生的缓存内容。实验所用 MS 数据来源于 SAGECaL 自带的测试数据。

2.2 时效性和合理性

本次实验在容器云环境下通过在不同数量服务器上部署不同规模的集群来观察集群部署和集群计算的时间变化情况。每组部署共进行五次并记录其完成时间，取其平均数作为最终部署完成时间。集群计算命令如下：

```
/usr/bin/time -v -o /root/project/ResultOutput/TimeProcessOutput mpirun                  \
--allow-run-as-root -np 32 -hostfile /root/hosts --mca yield_when_idle 1                 \
-mca orte_tmpdir_base /tmp /root/sagecal/MPI/sagecal-mpi                                  \
-f /root/data/MS/sm.ms -A 20 -P 2 -r 5 -s /root/sagecal/test/3c196.sky.txt               \
-c /root/sagecal/test/3c196.sky.txt.cluster -p /root/project/ResultOutput/sm.ms.solutions \
-n 16 -t 1 -e 3 -g 2 -l 10 -m 7 -x 10 -F 1 -j 5 > /root/project/ResultOutput/SAGECaLMPIProcessOutput
```

为了降低计算量，各组实验均只计算一个 MS 数据，ADMM 迭代次数设置为 20 次。进程数依据容器数来设定，每一个节点就是一个容器，一个进程对应一个容器。命令执行时间存放在/root/project/ResultOutput/目录下的 TimeProcessOutput 文件中。集群的缓存文件存放在/tmp 文件夹下。待处理的 MS 数据存放在/root/data/MS/文件夹下。集群计算选择使用 3c196 模式进行计算，最终计算结果存放在/root/project/ResultOutput/目录下的 sm.ms.solutions 文件中。

曙光天阔 620R 服务器配备两颗 4 核 CPU 和 4G 内存，总共有 12 台可用服务器。每一个集群节点在计算时启动一个进程，也就是在 96 个集群节点以内的规模情况下一个进程对应一个 CPU 内核。集群计算时间指集群部署好后从启动命令开始到得到计算结果的时间。整体部署时间指从私有镜像仓库下载镜像到集群部署完毕的时间。集群创建时间指服务器已经下载好镜像的情况下部署出一定规模的集群所花费的时间。

表 1 实验结果统计表

Tab.1 Statistical table of experimental results

服务器数	集群节点数	集群计算时间(秒)	整体部署时间(秒)	集群创建时间(秒)	部署与计算时间比(秒)
4	32	619.9	13.8	3.1	2.23%
8	64	501.3	24.1	4.8	4.81%
12	96	379.7	36.9	6.0	9.72%
12	500	394.2	39.3	9.2	9.97%
12	1000	416.0	43.8	12.9	10.53%

从表 1 可以看出集群的创建时间很短，加上 SAGECaL 镜像的传输时间，1000 个节点规模的集群在一分钟以内部署完毕是传统人工部署所不能比拟的。随着服务器数量的增加，计算能力得以提升，集群计算时间逐渐缩短。但随着计算进程数超过可用 CPU 内核数后，由于存在大量进程切换，计算时间有小幅增长。

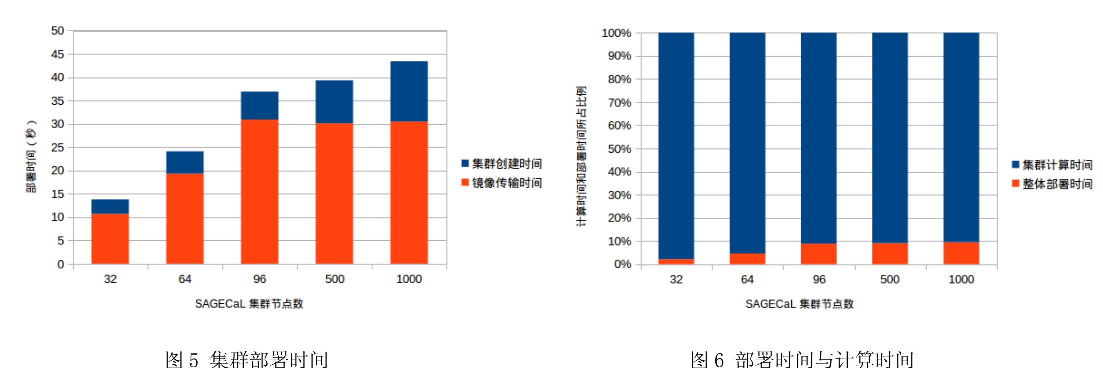


图 5 集群部署时间

Fig.5 Cluster delployment time

图 6 部署时间与计算时间

Fig.6 Deployment time & calculation time

如图 5 所示，整个部署过程中镜像的网络传输时间所占比例仍然很大，在镜像分发过程中集群的网络传输时间损耗依旧明显。但从图 6 可以看出在 1000 个节点的规模下整体自动部署时间与集群计算时间相比整体自动部署时间所占比例仅为 10%。由此可知，部署过程的时间损耗并不是主要因素，性能瓶颈仍然还在硬件计算能力上。通过分布式整合额外的计算资源并通过自动部署方法提高部署效率，在容器云下部署 SAGECaL 这类计算密集型的应用是合理的。

3 结论与展望

云环境中使用基于容器技术的自动部署方法在提高部署效率的同时便捷地整合了更多的计算资源，使得天文应用软件一次编译后便可以方便地运行在不同的硬件资源上。实验结果表明，本文给出的自动部署方法极大地提高了集群的部署效率，方便用户把主要精力集中在天文数据处理层面。

为了应对海量数据处理需求，在当前并行化发展的趋势下，部分天文应用软件已经对基于 MPI 进行分布式并行计算的并行方式提供了支持。基于容器技术的自动部署方法可以对所有基于 MPI 进行分布式并行计算的天文应用软件提供良好支持，是一种通用自动部署方法。

当下的 MPI 集群自动部署方法存在主节点一旦没有最先启动或是崩溃后整个部署过程将完全失败的问题。部署过程中虽然可以使用资源调度算法分配节点所在位置，但这样的资源调度方式仅适用于平等地位的节点调度，对于 MPI 集群这样有主次之分的节点调度还需要进一步完善调度机制。下一步将尝试使用 Zookeeper 或是 P2P 算法结合对物理机的实时性能动态采集与对比，探寻一种同时具备节点容错和自我演化功能的自动部署方法，让各节点初始状态下进行平等地位部署，然后根据自身所在环境性能情况和部署状况自我演化出合适的角色并进行动态自动部署。

致谢

感谢国家天文台-阿里云天文大数据联合研究中心对本项工作的支持。

参考文献

1. 陈泰燃，王威，王锋，等. 基于 MPI 的高性能 UVFITS 数据合成研究与应用[J]. 天文研究与技术，2016，13(2):184-189.

Chen Tairan, Wang Wei, Wang Feng, et al. The Study and Application of a High Performance UVFITS Assembly System Based on MPI[J]. Astronomical Research &

Technology——Publications of National Astronomical Observatories of China, 2016, 13(2):184-189.
2. 石聪明，张晓丽，等.MUSER 的负数据库接口设计与实现[J].天文研究与技术，2018,15(02):169-175.
Shi Congming, Zhang Xiaoli, et al.Design and Implementation of the MUSER Negative Database Interfaces[J].Astronomical Research & Technology——Publications of National Astronomical Observatories of China, 2018, 15(02):169-175.
3. Morris, D., et al., *Use of Docker for deployment and testing of astronomy software.* Astronomy & Computing, 2017. **20**.
4. Young, M.D., S. Hayashi, and A. Gopu. *StarDock: shipping customized computing environments to the data.* in *Software and Cyberinfrastructure for Astronomy IV.* 2016.
5. Merkel, D., *Docker: lightweight Linux containers for consistent development and deployment.* 2014. **2014**(239).
6. Rosado, T. and J. Bernardino, *An overview of openstack architecture.* 2014: ACM. 366-367.
7. Wei, S., et al., *OpenCluster: A Flexible Distributed Computing Framework for Astronomical Data Processing.* Publications of the Astronomical Society of the Pacific, 2016. **129**(972): p. 024001.
8. Kazemi, S., et al., *Radio interferometric calibration using the SAGE algorithm.* Monthly Notices of the Royal Astronomical Society, 2011. **414**(2): p. 1656-1666.
9. Kumazaki, K., S. Yatawatta, and S. Zaroubi. *Performance of SAGECal calibration.* in *General Assembly and Scientific Symposium.* 2014.
10. Naik, N. *Building a virtual system of systems using docker swarm in multiple clouds.* in *IEEE International Symposium on Systems Engineering.* 2016.
11. Nguyen, N. and D. Bein. *Distributed MPI cluster with Docker Swarm mode.* in *Computing and Communication Workshop and Conference.* 2017.

Automatic deployment method of astronomical application software based on container technology

Yao Kun[1] Dai Wei[1*] Yang Qiuping[1] Mei Ying[2,3] Shi Congming[1] Wang Feng[1,2,3]
(1 Computer Technology Application Key Lab of Yunnan Province, Kunming University of Science and Technology, Kunming 650500, China;
2 Yunnan Observatories of Chinese Academy of Sciences, Kunming 650011, China;
3 Center for Astrophysics, Guangzhou University, Guangzhou 510006, China;)

Abstract: The Square Kilometer Array(SKA) is under construction. And each sub-work package will also enter the critical design evaluation phase. The cloud-based and container-based technologies are the platform technologies that SKA Scientific Data Processor (SDP) may adopt in the future. This paper is aimed at the rapid deployment,

operation and measurement requirements of astronomical application software faced by SDP ultra-large-scale massive data processing. It fully considers the complex operation environment of astronomical application software and the difficulty in deploying ultra-large-scale computing clusters in cloud computing environment. For this reason, a general automatic deployment method for astronomical application using Docker technology is systematically studied and presented. The paper takes SAGECaL, a calibration software commonly used for visibility function as an example. Firstly, the characteristics of SAGECaL and the difficulties in distributed deployment are analyzed. The automatic deployment method of SAGECaL distributed cluster based on Docker container technology is given. The experimental results show that the automatic deployment method proposed in this paper greatly improves the deployment efficiency of SAGECaL distributed cluster, and meets the needs of the project team to undertake the deployment and switching of the basic platform required for SKA-SDP related testing. At the same time, this work also provides useful ideas for the rapid deployment and implementation of other astronomical software in the cloud.

Key words: Automatic deployment; Container technology; SAGECaL; SKA; SDP